Operating Systems - Scheduling

Project #4

Seong-Yeop Jeong

2020.04.21.

Reminder) Late Submission policy

- You can use up to 5 slip days for this semester
  - You should explicitly declare the number of slip days to use in the Q&A board on the submission server
  - https://sys.snu.ac.kr/main.php?classIdx=1&menu=Board
- 25% penalty per day after slip day

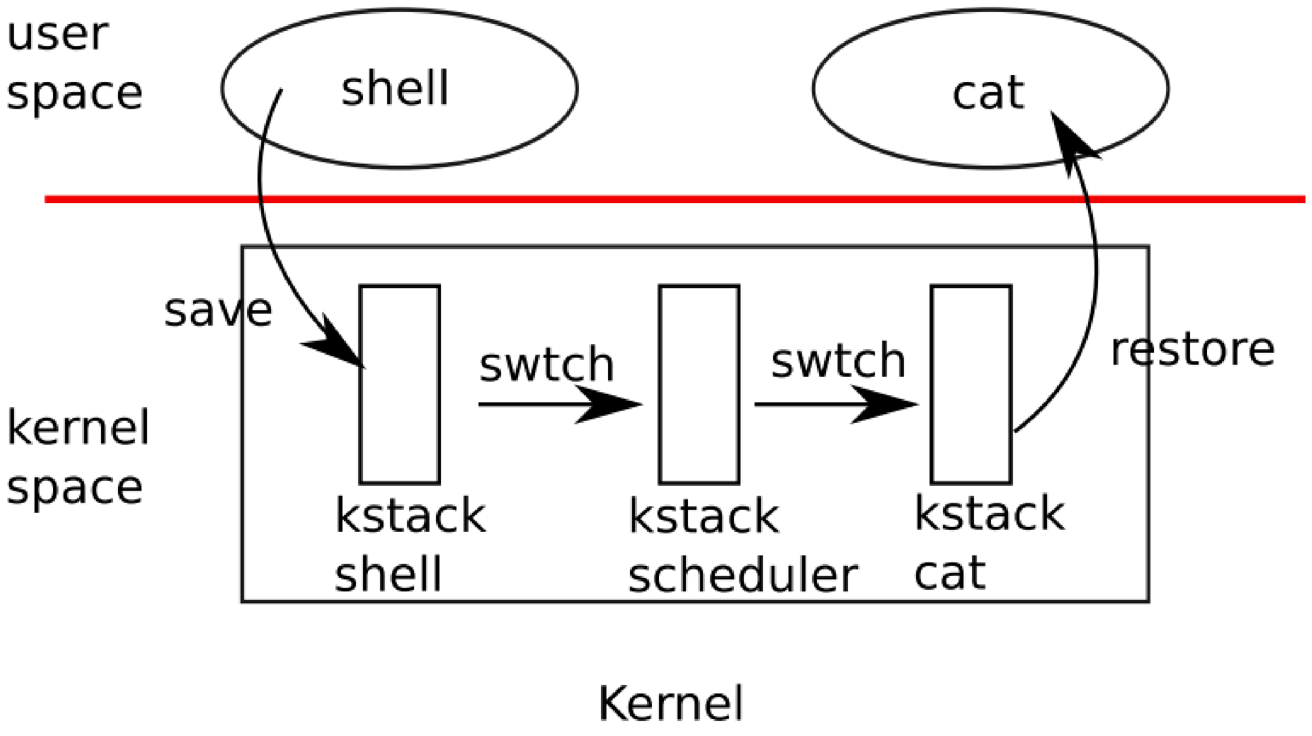
Xv6 Process

- Process states (in proc.h)
  - Enum prostate { UNUSED, SLEEPING, RUNNABLE, RUNNING, ZOMBIE }

- UNUSED : Not used
- SLEEPING : Wait for I/O, wait(), or sleep()
- RUNNABLE : Ready to run
- RUNNING : Running on CPU
- ZOMBIE : Exited, wait for parent call wait()

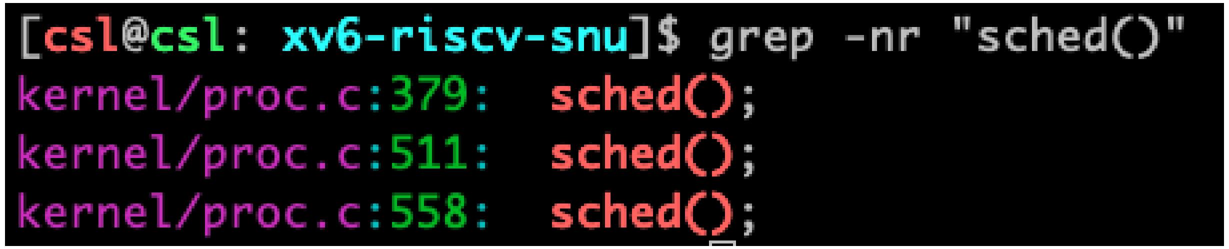
Xv6 Scheduler

- Switching from one user process to another
  - In this example, xv6 runs with one CPU (One scheduler thread)

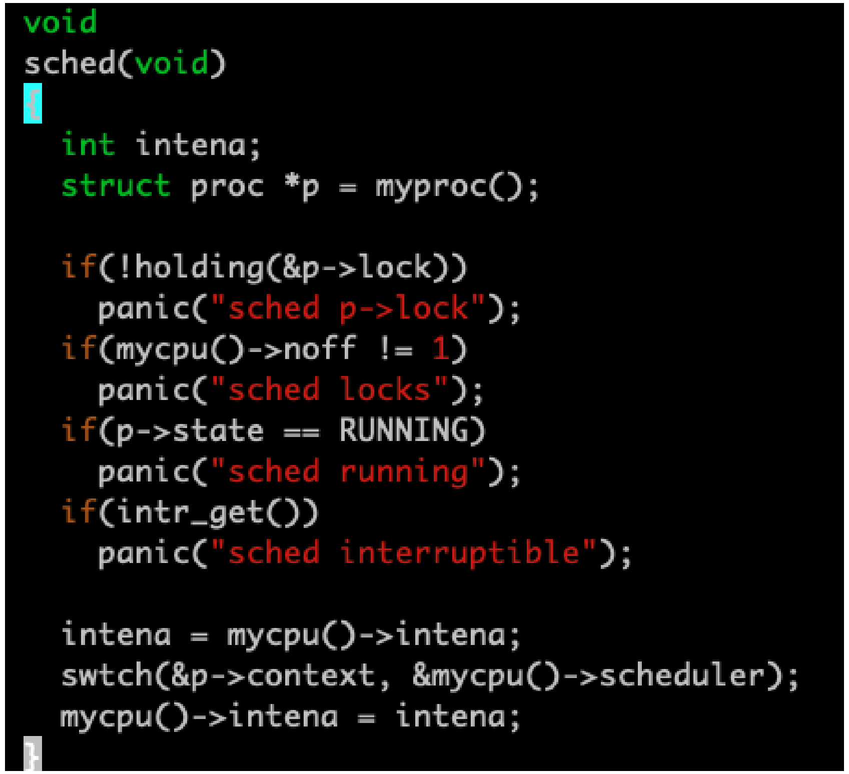


Entering scheduler

- Entering scheduler when
  1. Exiting process
  2. Sleeping process
  3. Yielding CPU (timer interrupt)



```
[csl@csl: xv6-riscv-snu]$ grep -nr "sched()"
kernel/proc.c:379:   sched();
kernel/proc.c:511:   sched();
kernel/proc.c:558:   sched();
```



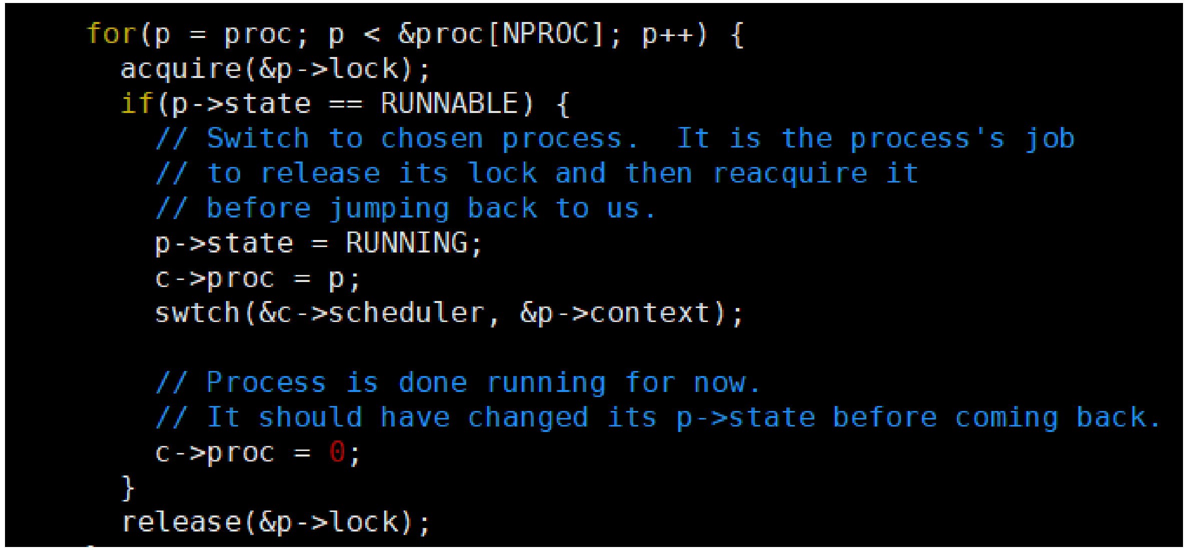
```
void
sched(void)
{
  int intena;
  struct proc *p = myproc();

  if(!holding(&p->lock))
    panic("sched p->lock");
  if(mycpu()->noff != 1)
    panic("sched locks");
  if(p->state == RUNNING)
    panic("sched running");
  if(intr_get())
    panic("sched interruptible");

  intena = mycpu()->intena;
  swtch(&p->context, &mycpu()->scheduler);
  mycpu()->intena = intena;
}
```

Xv6 Scheduler

- A simple scheduling policy, which runs each process in turn.
  - This policy is called round robin
- Each CPU calls scheduler() after setting itself up.
- Scheduler never returns. It loops, doing
  - choose a process to run.
  - swtch to start running that process.
  - eventually that process transfers control
  - via swtch back to the scheduler
- in kernel/proc.c
  - void scheduler(void)

Xv6 Scheduler in proc.c

- The scheduler loops over the process table looking for a p-> state == RUNNABLE
- Once it finds a process, it sets the per-CPU current process(c->proc)
- Marks the process as RUNNING, and then calls swtch to start running



```
for(p = proc; p < &proc[NPROC]; p++) {
  acquire(&p->lock);
  if(p->state == RUNNABLE) {
    // Switch to chosen process.  It is the process's job
    // to release its lock and then reacquire it
    // before jumping back to us.
    p->state = RUNNING;
    c->proc = p;
    swtch(&c->scheduler, &p->context);

    // Process is done running for now.
    // It should have changed its p->state before coming back.
    c->proc = 0;
  }
  release(&p->lock);
```

Timer interrupt in trap.c

- RISC-V has three modes in which the CPU can execute instructions machine mode, supervisor mode, and user mode

- Time interrupt in xv6 is machine mode.

- The yield calls in usertrap and kerneltrap cause this switching

Project#4 – Linux 2.4 Scheduler

- In this project, you have to
  - 1. Implement the nice() system call
  - 2. Implement a Simplified Linux 2.4 Scheduler
  - 3. Revise the getticks() system call

- Due date is May 3 (Sunday) 11:59PM

nice system call - 1

- int nice(int pid, int inc)
  - Add inc to the current nice value of the process pid.
  - The range of nice value is -20 to 19
  - A higher nice value means a lower priority
  - When a process is created, its nice value is the same as the parent's
  - The nice value of the init process is zero
  - if pid is positive, then the nice value of process with the specified pid is changed.
  - If pid is zero, then the nice value of the calling process if is changed.

nice system call - 2

- return value
  - On success, zero is returned.
  - On error, -1 is returned.
    - pid is negative.
    - There is no valid process that has pid.
    - The resulting nice value exceeds the range [-20, 19]

Simplified Linux 2.4 scheduler - 1

- On every timer ticks, the $p \rightarrow counter$ value is decremented by 1
  - When $p \rightarrow counter$ is 0, the process used up all its time slice
  - and the scheduler is invoked !
- When the scheduler is invoked, the scheduler picks the process that has maximum goodness value.

```
goodness(p) = 0                                 ,if p→counter == 0
            = p→counter  + (20 - p→nice)   ,otherwise
```

Simplified Linux 2.4 scheduler - 2

- If there are no runnable processes, the scheduler busy-waits in current epoch
- When the $p \rightarrow counter$ value of all the runnable process will become 0,
  - The scheduler start new epoch, and reset the time slice for all processes

For runnable processes

$$p{\rightarrow}counter = ((20 - (p{\rightarrow}nice)) >> 2) + 1$$

For blocked processes (when $p{\rightarrow}counter$ is not 0)

$$p{\rightarrow}counter = (p{\rightarrow}counter >> 1) + ((20 - (p{\rightarrow}nice)) >> 2) + 1$$

Simplified Linux 2.4 scheduler - 3

- When fork is executed, the process' remaining time slice is split
  - parent : $p \rightarrow counter \ >> \ 1$
  - child : $(p \rightarrow counter \ + 1) \ >> \ 1$

- Once scheduled, the running process is not preempted until the end of its time slice

Revise getticks system call - 1

- int getticks(int pid)
  - The getticks system call returns the number of ticks used by the process pid
  - The skeleton code already includes an implementation of the getticks() system call for the default round-robin scheduler.

  - Pre-implemented getticks system call doesn’t return the correct number of ticks
  - So, you revise it

getticks system call - 2

- int getticks(int pid)
  - If pid is positive, return the number of ticks used by the process.
  - If pid is zero, return the number of ticks used by the calling process.

- return value
  - On success, the number of ticks is returned.
  - On error, -1 is returned.

Project #4 – submission

- In this project, you have to submit a report explaining your implementation. (+ source code)

- These must be included in your report.
  - How to implement scheduling algorithm
    - You must explain details about your schedule
  - How to make sure the getticks() system call returns the correct number of ticks
  - Schedtest2 result (make qemu-log result, xv6.log) and an explanation
  - Schedtest2 image (make png result)

Project #4 – restrictions

- The CPUS is already set to 1 in the Makefile
- Your implementation should pass the following test programs available on xv6
  - usertests
  - schedtest1
  - Schedtest2

- Do not add any system calls other than nice() and getticks()
- You only need to modify those files in the ./kernel directory

You may want to see…

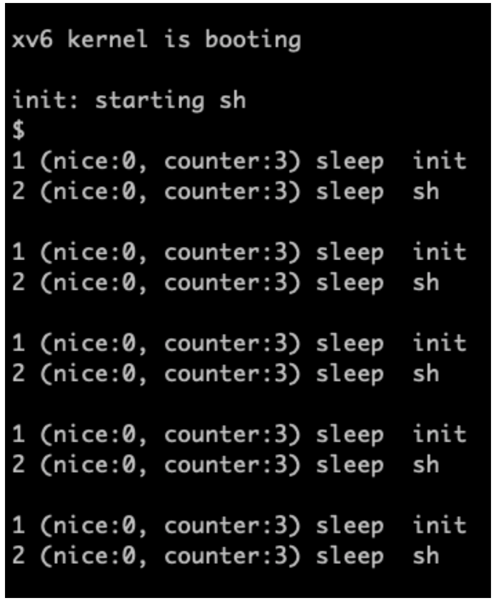
- modification
  - proc.c
  - trap.c

- In xv6 book
  - Chapter 4.1 ~ 4.4 (about trap)
  - Chapter 6 (about scheduling)

Example - boot

$p{\rightarrow}counter = ((20 - (p{\rightarrow}nice)) \gg 2) + 1$

- The init process' nice is zero, which is the default value for $p{\rightarrow}nice$, and the counter is six according to the formula above.
- When you first boot up, $p{\rightarrow}nice$ should be set to zero, and the counter should be parent, child all 3 because it was inherited from init.
- And because there is no runnable process, do not enter the new epoch and do not reassign the counter.



```
xv6 kernel is booting

init: starting sh
$
1 (nice:0, counter:3) sleep  init
2 (nice:0, counter:3) sleep  sh

1 (nice:0, counter:3) sleep  init
2 (nice:0, counter:3) sleep  sh

1 (nice:0, counter:3) sleep  init
2 (nice:0, counter:3) sleep  sh

1 (nice:0, counter:3) sleep  init
2 (nice:0, counter:3) sleep  sh

1 (nice:0, counter:3) sleep  init
2 (nice:0, counter:3) sleep  sh
```

Busy waiting

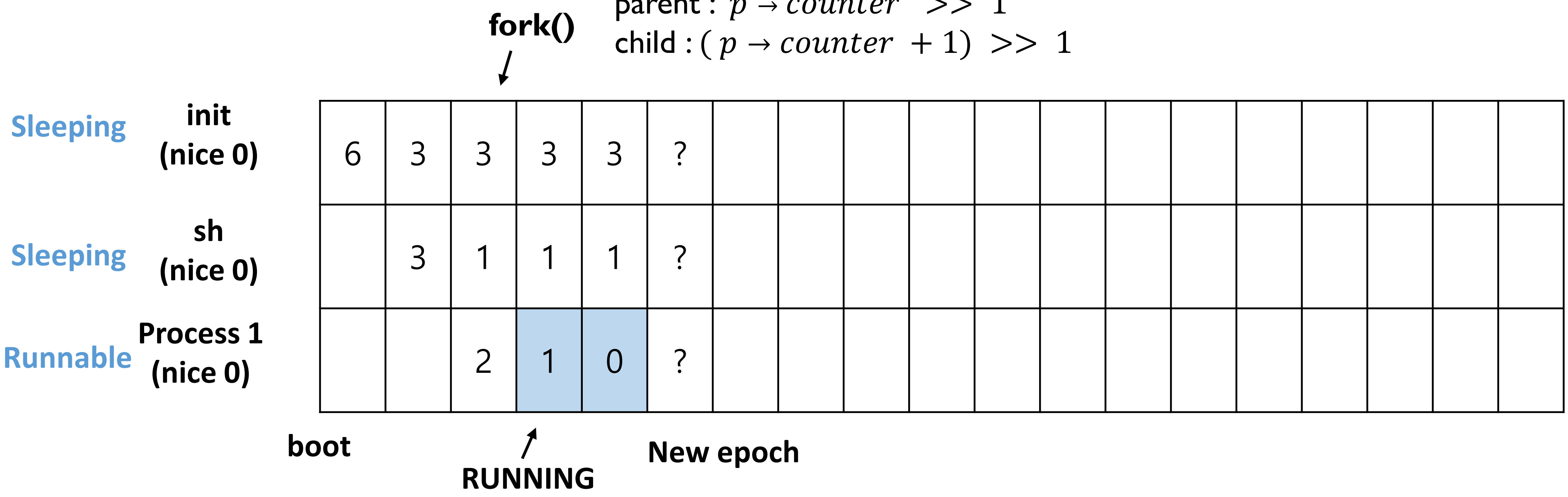
fork()

parent : $p \rightarrow counter \;>>\; 1$
child : $(p \rightarrow counter + 1) \;>>\; 1$

Sleeping	**init (nice 0)**	6	3																	
Sleeping	**sh (nice 0)**		3																	

boot **Busy waiting**

Blocked process



fork()

parent : $p \rightarrow counter \;>>\; 1$

child : $(p \rightarrow counter \;+ 1) \;>>\; 1$

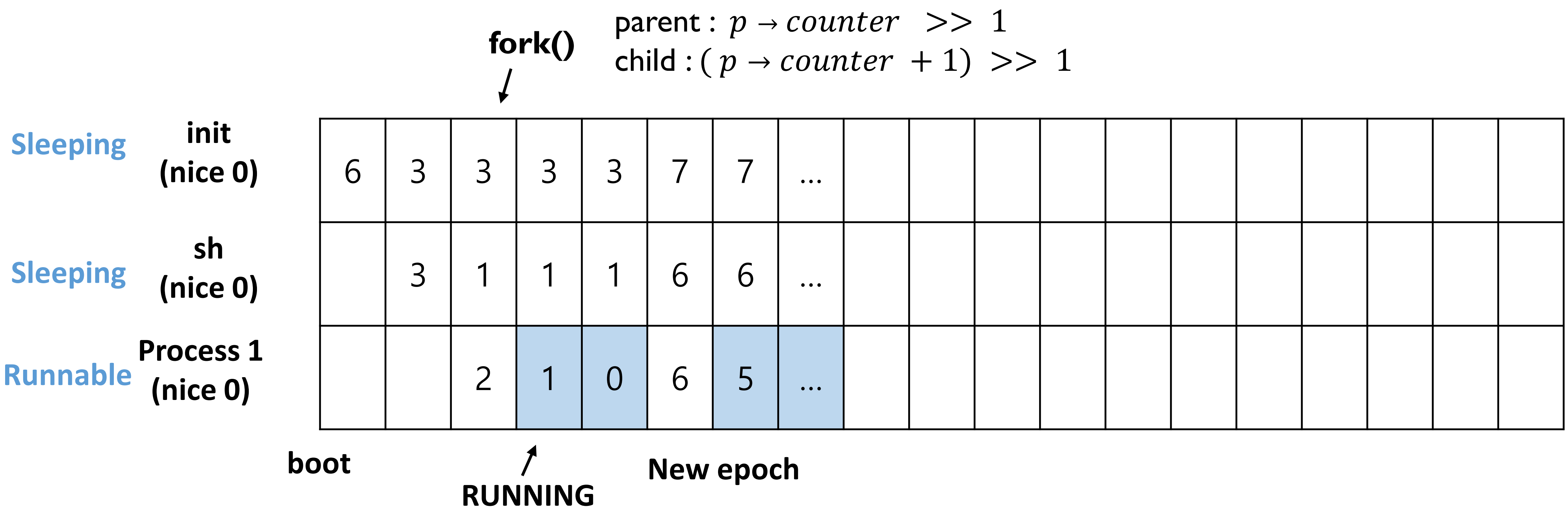
Sleeping	init (nice 0)	6	3	3	3	3	?													
Sleeping	sh (nice 0)		3	1	1	1	?													
Runnable	Process 1 (nice 0)			2	1	0	?													

boot

RUNNING

New epoch

Blocked process



parent : $p \rightarrow counter \;\; >> \; 1$
child : $(p \rightarrow counter \; + 1) \;\; >> \; 1$

fork()

Sleeping	init (nice 0)	6	3	3	3	3	7	7	…											
Sleeping	sh (nice 0)		3	1	1	1	6	6	…											
Runnable	Process 1 (nice 0)			2	1	0	6	5	…											

boot

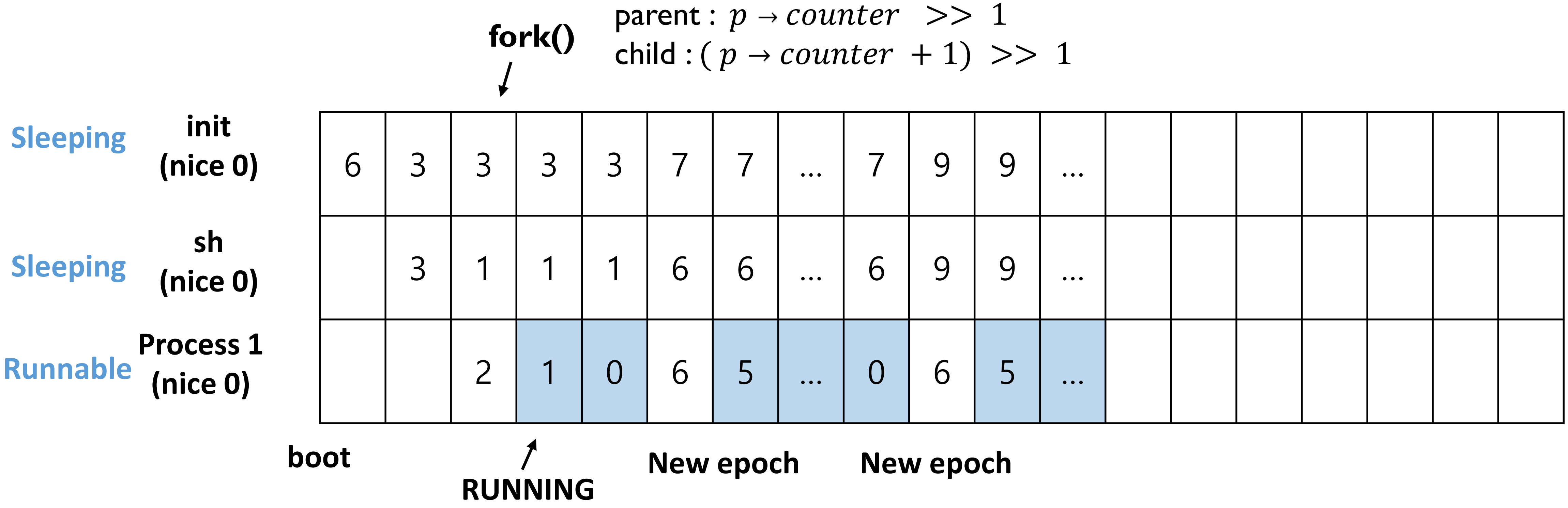
RUNNING

New epoch

For runnable process $p{\rightarrow}counter$ = `((20 - (`$p{\rightarrow}nice$`)) >> 2) + 1`

For blocked process $p{\rightarrow}counter$ = `(`$p{\rightarrow}counter$` >> 1) +((20 - (`$p{\rightarrow}nice$`)) >> 2) + 1`

Blocked process



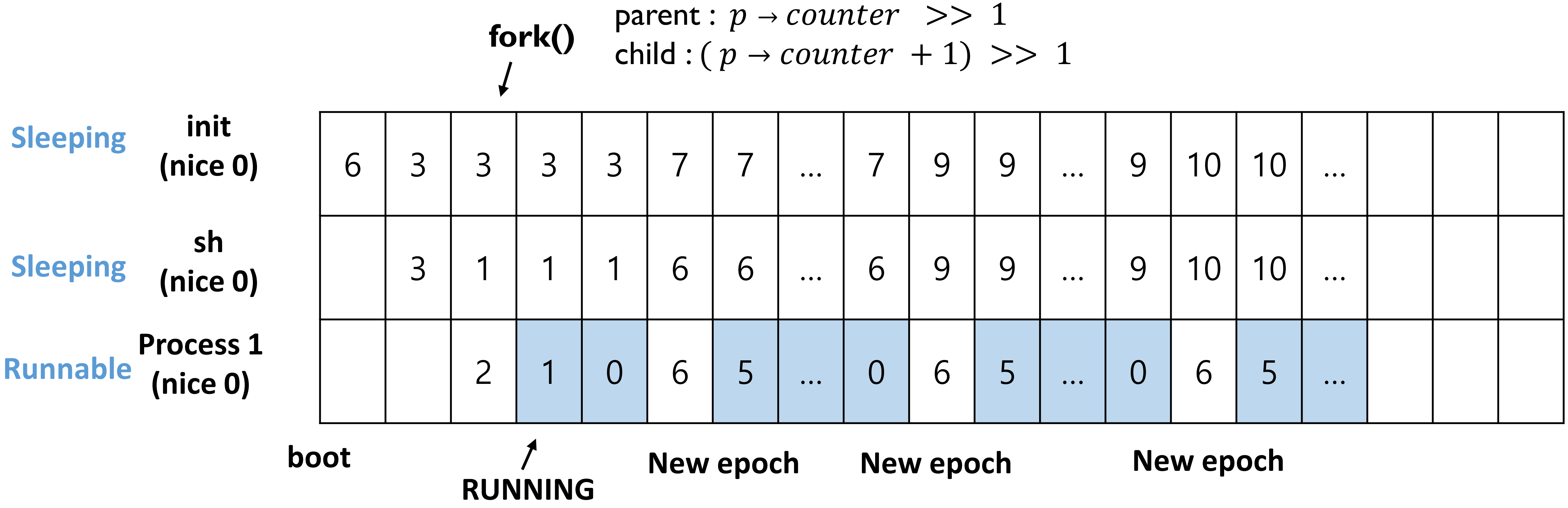
parent : $p \to counter \; >> \; 1$
child : $(p \to counter \; + 1) \; >> \; 1$

Sleeping	init (nice 0)	6	3	3	3	3	7	7	…	7	9	9	…							
Sleeping	sh (nice 0)		3	1	1	1	6	6	…	6	9	9	…							
Runnable	Process 1 (nice 0)			2	1	0	6	5	…	0	6	5	…							

For runnable process $p{\to}counter = ((20 - (p{\to}nice)) >> 2) + 1$

For blocked process $p{\to}counter = (p{\to}counter >> 1) + ((20 - (p{\to}nice)) >> 2) + 1$

Blocked process



fork()

parent : $p \rightarrow counter \;>>\; 1$

child : $(p \rightarrow counter + 1) \;>>\; 1$

Sleeping	init (nice 0)	6	3	3	3	3	7	7	...	7	9	9	...	9	10	10	...			
Sleeping	sh (nice 0)		3	1	1	1	6	6	...	6	9	9	...	9	10	10	...			
Runnable	Process 1 (nice 0)			2	1	0	6	5	...	0	6	5	...	0	6	5	...			

boot

RUNNING

New epoch

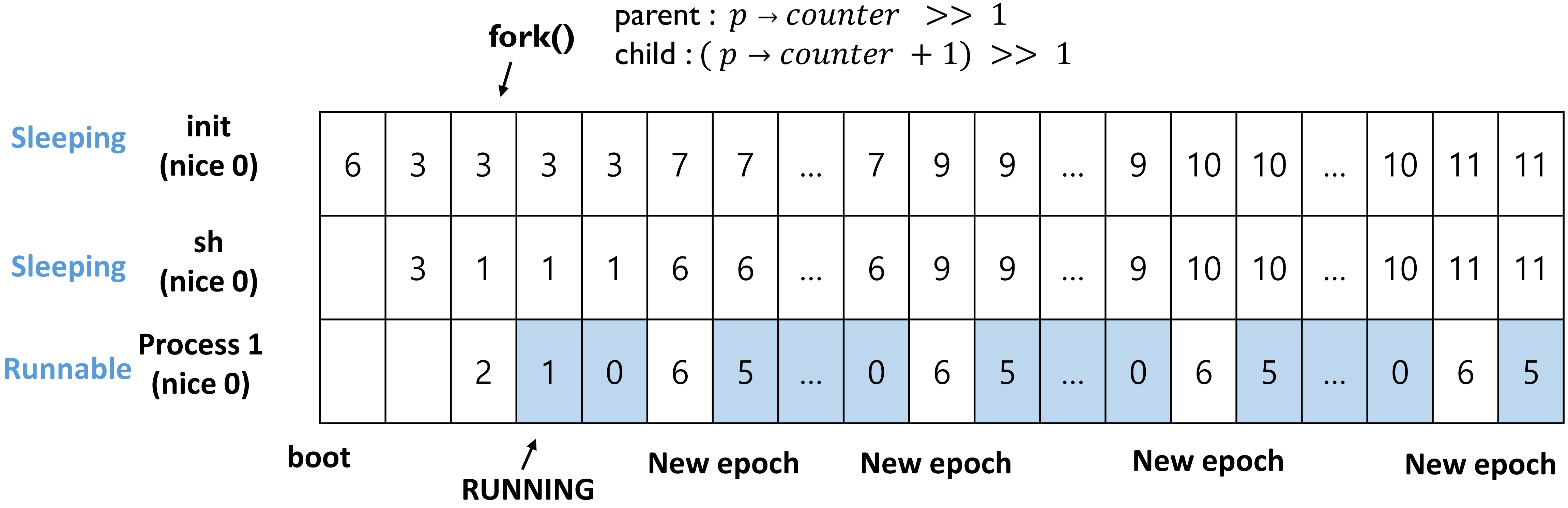
New epoch

New epoch

For runnable process $p{\rightarrow}counter = ((20 - (p{\rightarrow}nice)) >> 2) + 1$

For blocked process $p{\rightarrow}counter = (p{\rightarrow}counter >> 1) + ((20 - (p{\rightarrow}nice)) >> 2) + 1$

Blocked process



fork()

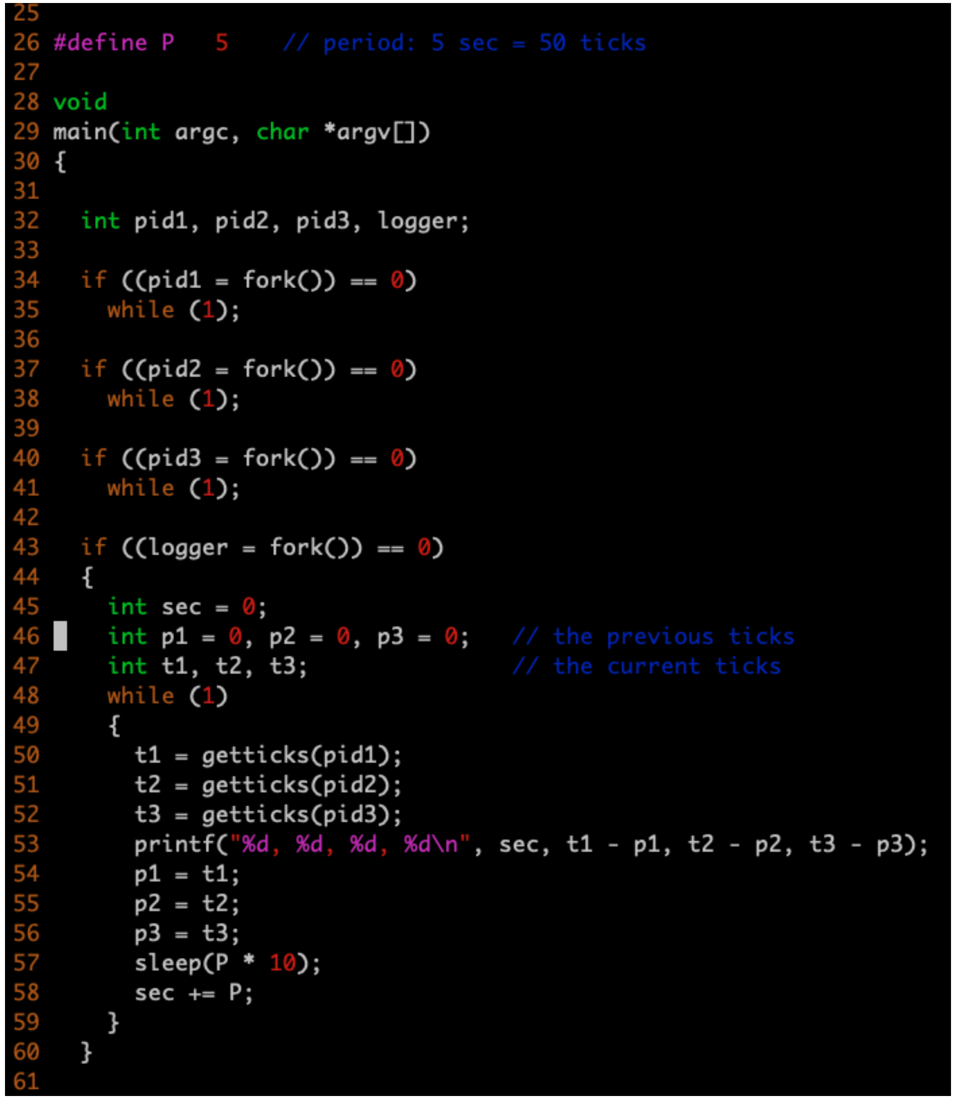
parent : $p \rightarrow counter \;\; >> \; 1$
child : $(p \rightarrow counter \; + 1) \;\; >> \; 1$

Sleeping	init (nice 0)	6	3	3	3	3	7	7	…	7	9	9	…	9	10	10	…	10	11	11
Sleeping	sh (nice 0)		3	1	1	1	6	6	…	6	9	9	…	9	10	10	…	10	11	11
Runnable	Process 1 (nice 0)			2	1	0	6	5	…	0	6	5	…	0	6	5	…	0	6	5

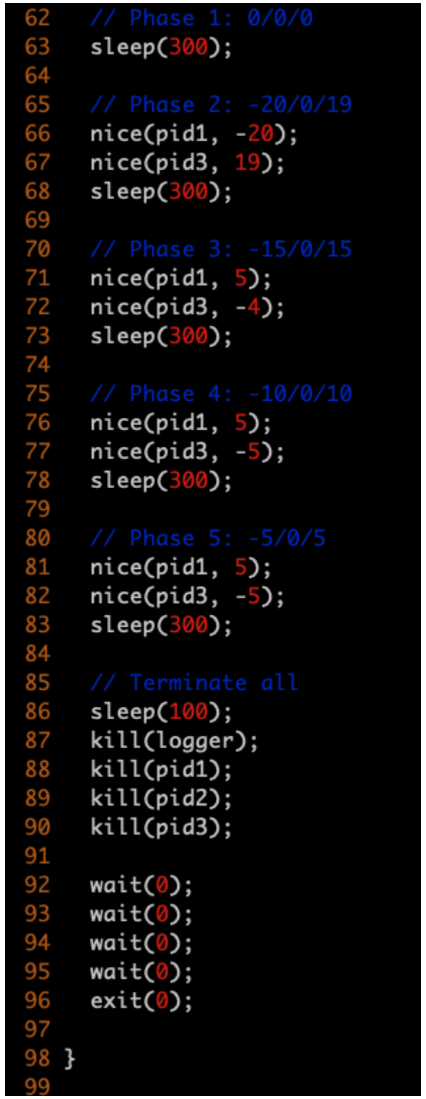
For runnable process $p \rightarrow counter$ = `((20 - (`$p \rightarrow nice$`)) >> 2) + 1`

For blocked process $p \rightarrow counter$ = `(`$p \rightarrow counter$` >> 1) +((20 - (`$p \rightarrow nice$`)) >> 2) + 1`

Schedtest2



```c

#define P   5    // period: 5 sec = 50 ticks

void
main(int argc, char *argv[])
{

  int pid1, pid2, pid3, logger;

  if ((pid1 = fork()) == 0)
    while (1);

  if ((pid2 = fork()) == 0)
    while (1);

  if ((pid3 = fork()) == 0)
    while (1);

  if ((logger = fork()) == 0)
  {
    int sec = 0;
    int p1 = 0, p2 = 0, p3 = 0;   // the previous ticks
    int t1, t2, t3;               // the current ticks
    while (1)
    {
      t1 = getticks(pid1);
      t2 = getticks(pid2);
      t3 = getticks(pid3);
      printf("%d, %d, %d, %d\n", sec, t1 - p1, t2 - p2, t3 - p3);
      p1 = t1;
      p2 = t2;
      p3 = t3;
      sleep(P * 10);
      sec += P;
    }
  }

```

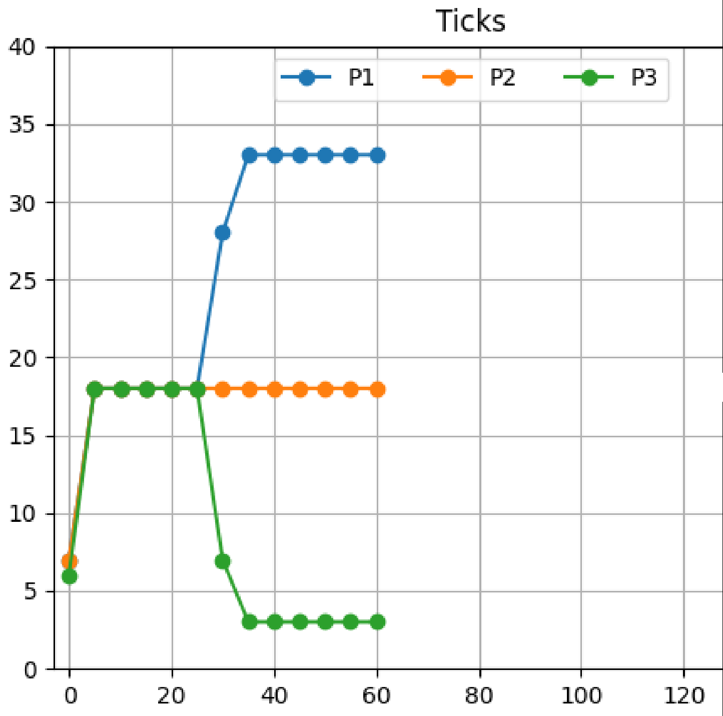


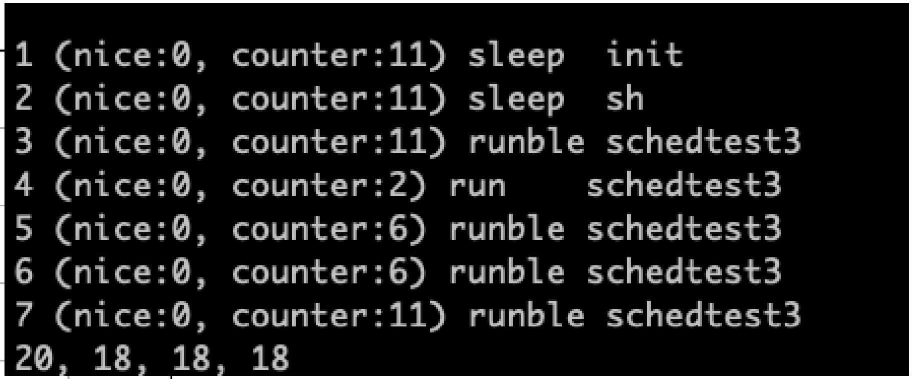
```c
  // Phase 1: 0/0/0
  sleep(300);

  // Phase 2: -20/0/19
  nice(pid1, -20);
  nice(pid3, 19);
  sleep(300);

  // Phase 3: -15/0/15
  nice(pid1, 5);
  nice(pid3, -4);
  sleep(300);

  // Phase 4: -10/0/10
  nice(pid1, 5);
  nice(pid3, -5);
  sleep(300);

  // Phase 5: -5/0/5
  nice(pid1, 5);
  nice(pid3, -5);
  sleep(300);

  // Terminate all
  sleep(100);
  kill(logger);
  kill(pid1);
  kill(pid2);
  kill(pid3);

  wait(0);
  wait(0);
  wait(0);
  wait(0);
  exit(0);

}

```

Example – schedtest2

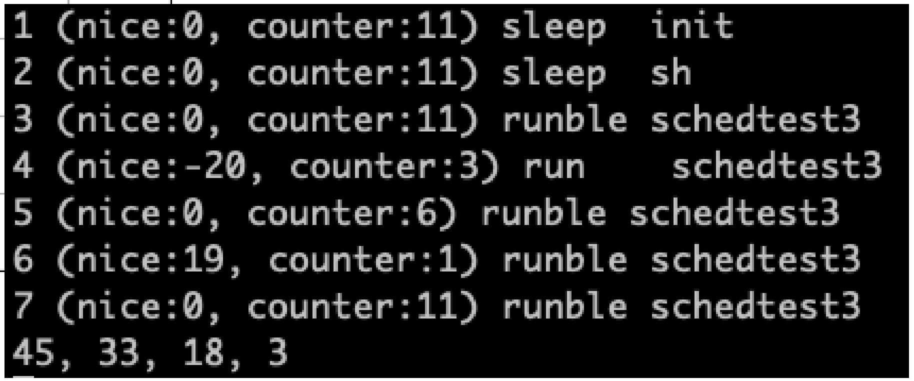
- If you have a process with a nice value of -20, 0,19 ,what order is it scheduled? (P1:0, P2:0, P3:0 -> P1: -20, P2:0, P3:19)

```
xv6 kernel is booting

init: starting sh
$ schedtest2
0, 7, 7, 6
5, 18, 18, 18
10, 18, 18, 18
15, 18, 18, 18
20, 18, 18, 18
25, 18, 18, 18
30, 28, 18, 7
35, 33, 18, 3
40, 33, 18, 3
45, 33, 18, 3
50, 33, 18, 3
55, 33, 18, 3
60, 33, 18, 3
```





```
1 (nice:0, counter:11) sleep  init
2 (nice:0, counter:11) sleep  sh
3 (nice:0, counter:11) runble schedtest3
4 (nice:0, counter:2) run     schedtest3
5 (nice:0, counter:6) runble schedtest3
6 (nice:0, counter:6) runble schedtest3
7 (nice:0, counter:11) runble schedtest3
20, 18, 18, 18
```



```
1 (nice:0, counter:11) sleep  init
2 (nice:0, counter:11) sleep  sh
3 (nice:0, counter:11) runble schedtest3
4 (nice:-20, counter:3) run    schedtest3
5 (nice:0, counter:6) runble schedtest3
6 (nice:19, counter:1) runble schedtest3
7 (nice:0, counter:11) runble schedtest3
45, 33, 18, 3
```

Phase 1 P1:0, P2:0, P3:0

Runnable

RUNNING

Process 1 (nice 0)	6	5	4	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Process 2 (nice 0)	6	6	6	6	6	6	6	5	4	3	2	1	0	0	0	0	0	0	0
Process 3 (nice 0)	6	6	6	6	6	6	6	6	6	6	6	6	6	5	4	3	2	1	0

new epoch

end epoch

For runnable process $p{\to}counter$ `= ((20 - (`$p{\to}nice$`)) >> 2) + 1`

Phase 2 P1: -20, P2:0, P3:19

Runnable

RUNNING

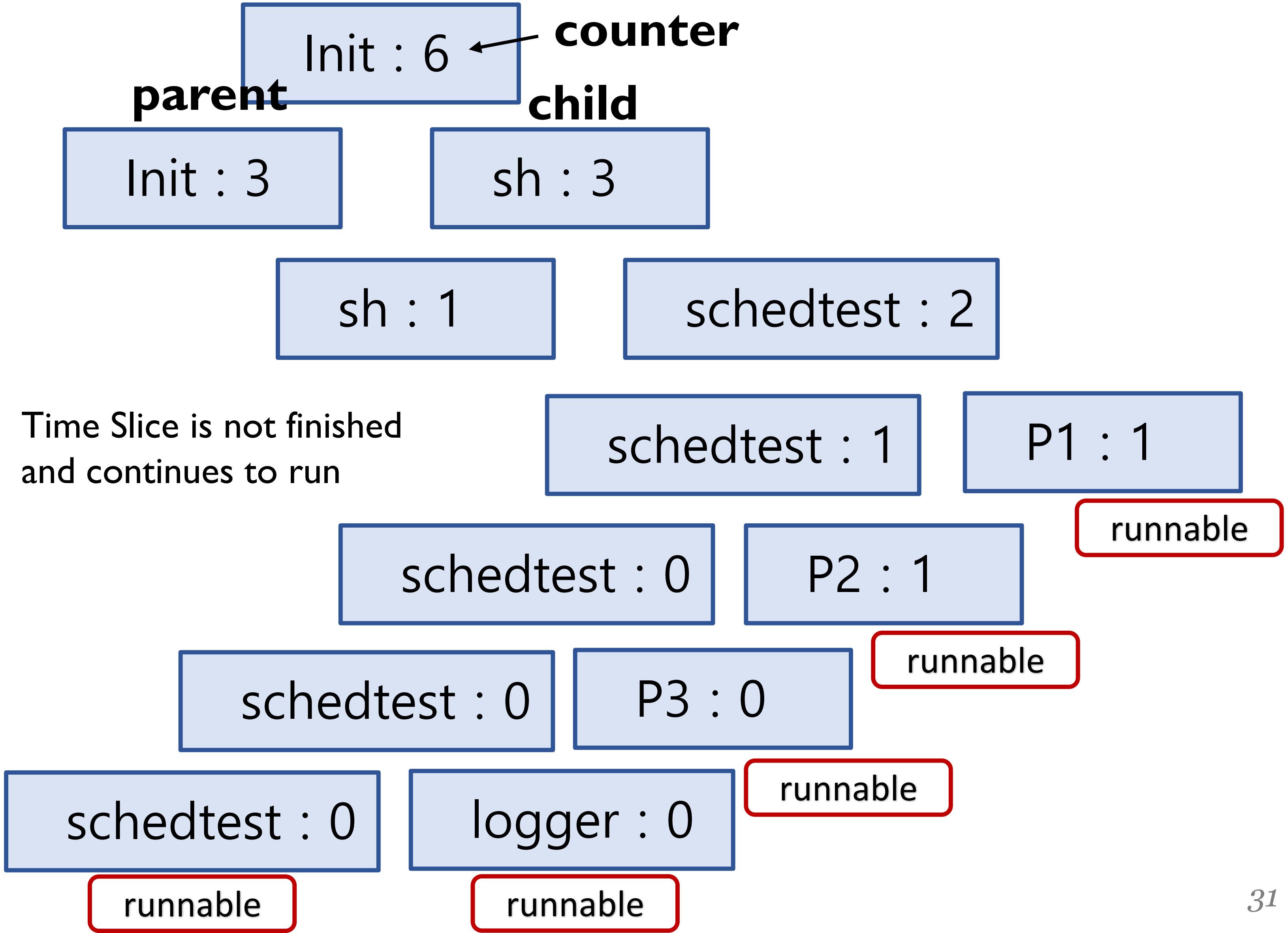
Process 1 (nice -20)	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0	0
Process 2 (nice 0)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Process 3 (nice 19)	6	6	6	6	6	6	6	6	6	6	6	6	5	4	3	2	1	0	0

new epoch **end epoch**

For runnable process $p \rightarrow counter$ = ((20 - ($p \rightarrow nice$)) >> 2) + 1

Why does the tick become like that at first?

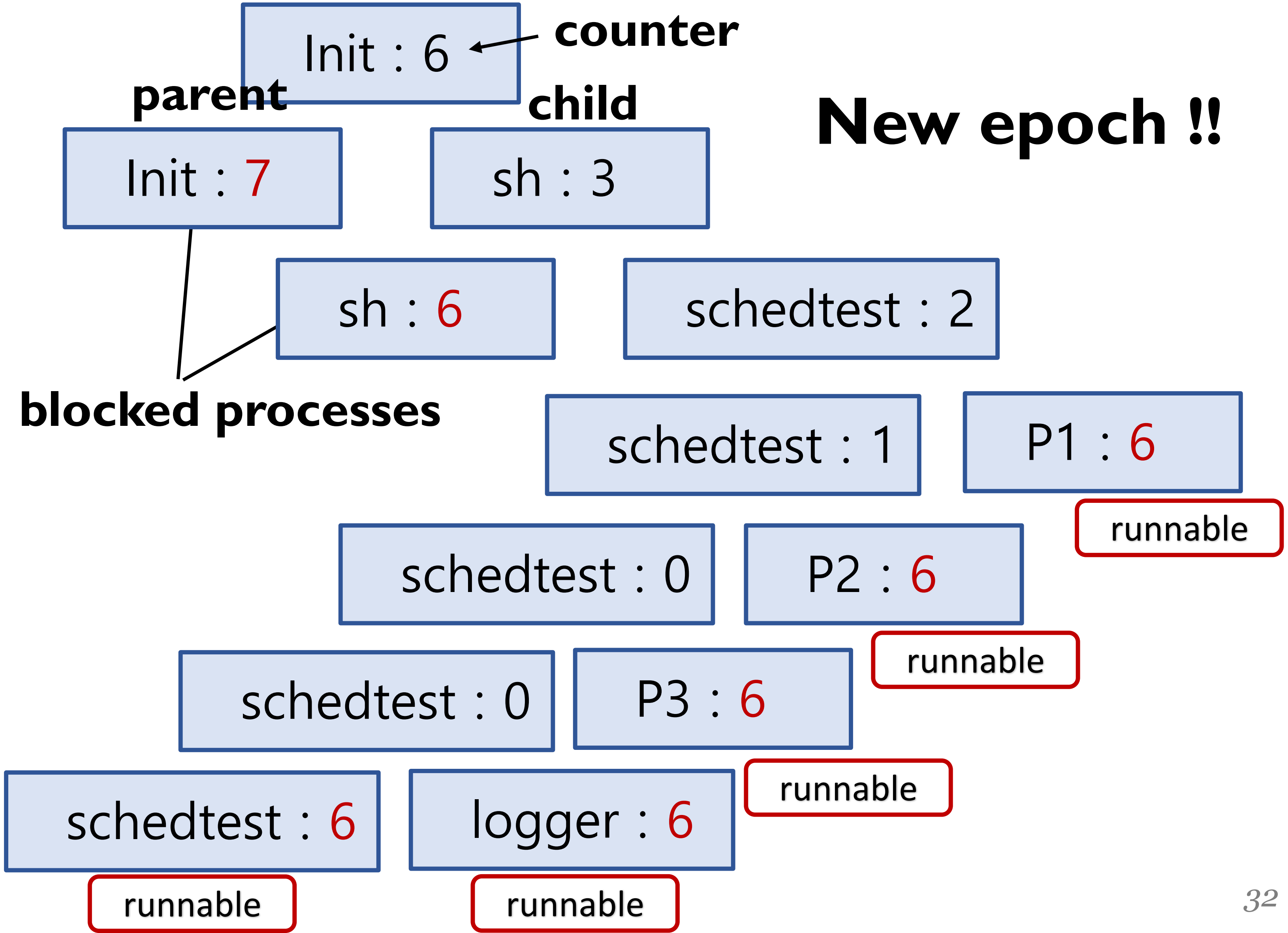
```
xv6 kernel is booting

init: starting sh
$ schedtest …
0, 7, 7, 6
5, 18, 18, 18
10, 18, 18, 18
15, 18, 18, 18
20, 18, 18, 18
25, 18, 18, 18
30, 28, 18, 7
35, 33, 18, 3
40, 33, 18, 3
45, 33, 18, 3
50, 33, 18, 3
55, 33, 18, 3
60, 33, 18, 3
```



Time Slice is not finished and continues to run

Why does the tick become like that at first?

```
xv6 kernel is booting

init: starting sh
$ schedtest …
0, 7(1+6), 7(1+6), 6(0+6)
5, 18, 18, 18
10, 18, 18, 18
15, 18, 18, 18
20, 18, 18, 18
25, 18, 18, 18
30, 28, 18, 7
35, 33, 18, 3
40, 33, 18, 3
45, 33, 18, 3
50, 33, 18, 3
55, 33, 18, 3
60, 33, 18, 3
```

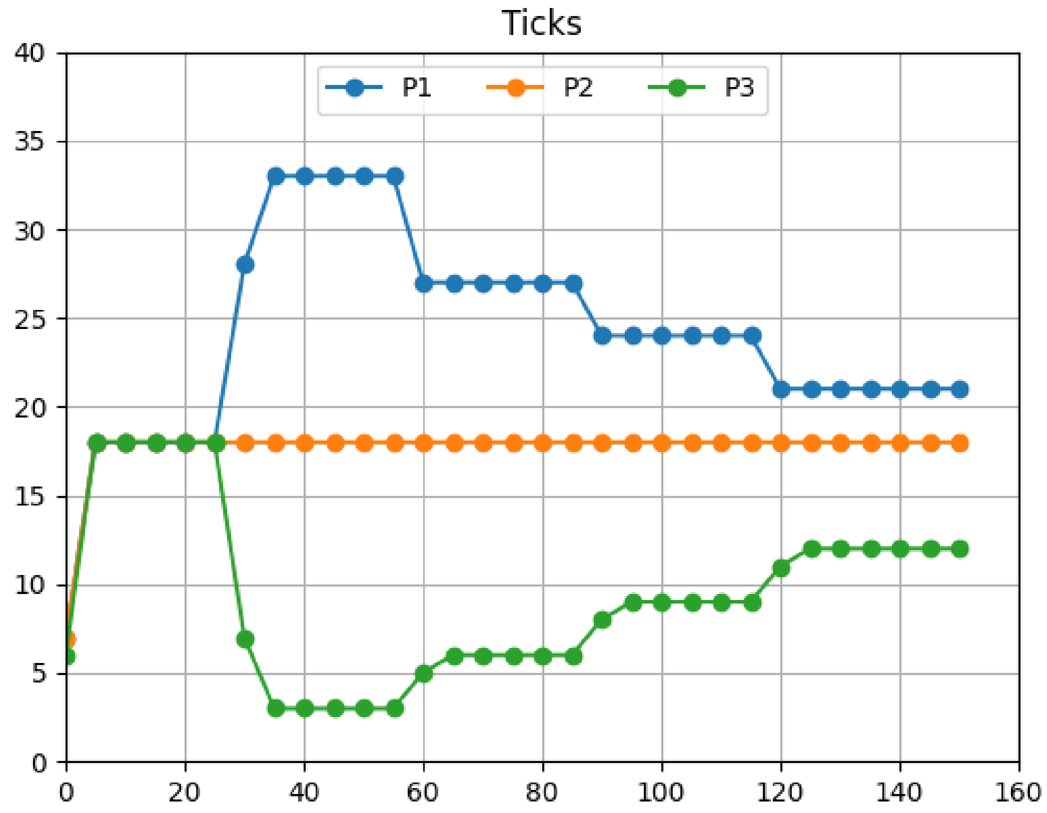


Example – schedtest2

```
xv6 kernel is booting

init: starting sh
$ schedtest2
0, 7, 7, 6
5, 18, 18, 18 -> phase1
10, 18, 18, 18
15, 18, 18, 18
20, 18, 18, 18
25, 18, 18, 18
30, 28, 18, 7
35, 33, 18, 3 -> phase2
40, 33, 18, 3
45, 33, 18, 3
50, 33, 18, 3
55, 33, 18, 3
60, 27, 18, 5
65, 27, 18, 6 -> phase3
70, 27, 18, 6
```

```
75, 27, 18, 6
80, 27, 18, 6
85, 27, 18, 6
90, 24, 18, 8
95, 24, 18, 9 -> phase4
100, 24, 18, 9
105, 24, 18, 9
110, 24, 18, 9
115, 24, 18, 9
120, 21, 18, 11
125, 21, 18, 12 -> phase5
130, 21, 18, 12
135, 21, 18, 12
140, 21, 18, 12
145, 21, 18, 12
150, 21, 18, 12
```

phase1	0/0/0
phase2	-20/0/19
phase3	-15/0/15
phase4	-10/0/10
phase5	-5/0/5



FAQ

- Even if you change the nice value to nice(), you do not need to change the p->counter immediately.
  - Just write the changed value when the p->nice value is used.
- What happens when there's a block in the middle ?
  - $p{\rightarrow}counter = (p{\rightarrow}counter >> 1) +((20 - (p{\rightarrow}nice)) >> 2) + 1$
- When timer interrupt is occurred, the p->counter value may become negative once the p->counter value is reduced.
- If runnable process is not available, continue waiting without starting new epoch

Thank you!

- Any questions?